Publications Internes de l'IRISA
ISSN : 2102-6327
PI 1983 – août 2011

Dealing with Asynchronous Observation Issues in Passive Interoperability Testing

Nanxing CHEN*
nanxing.chen@irisa.fr

Abstract: The purpose of interoperability testing is to ensure that interconnected protocol implementations communicate correctly while providing the expected services. To perform interoperability testing, conventional approaches rely on the active testing method, which stimulates the system under test to accomplish a diagnose. However, the arbitrary stimuli injected by using active testing method disturb inevitably the normal operation of the system under test. On the contrary, passive testing is a technique for the verification of behavioral properties of systems by only observing their external behavior. However, passive testing method often encounters observation problems due to the asynchronous test environment. In fact, the delay in such environment may influence the observation of the real order of the outputs produced by the implementations under test. Consequently, the verdict emitted by the test system maybe permissive or bias. As passive testing depends only on observation, issues here are then to keep a high level of observation in the testing activity, ensuring that no biased or permissive verdict is introduced. To solve the observation problems in passive interoperability testing, this report proposes a method based on logical clocks. Each output produced by the implementations under test is associated with a time stamp, which is carried on the fly during its transmission. By analyzing the time stamps, the tester is able to recreate the true sequence of the observed events so as to give a correct verdict.

Key-words: interoperability testing, time stamps, passive testing

---

Résolution des Problèmes d'observation Asynchrones dans le Test d'interoperabilité Passif

Résumé : *L'objectif du test d'interopérabilité est de s'assurer à la fois que les implémentations interagissent correctement et qu'elles rendent les services prévus dans leur spécification pendant leur interaction. Pour effectuer le test d'interopérabilité, les approches classiques s'appuient sur la méthode active, dont lŠobjectif est de tester les implémentations d'un système en pratiquant une suite de contrôles et d'observations sur celui-ci. Pourtant, les stimuli injectés en utilisant la méthode active perturbent inévitablement les opérations normales du système à tester. Au contraire, le test passif a pour objectif de détecter des erreurs dans un système en observant uniquement ses comportements. Cependant, la méthode passive rencontre souvent des difficultés d'observation en raison de la nature asynchrone des systèmes distribués. En effet, les délais dŠacquisition des donnés dans lŠenvironnement du test peuvent perturber l'ordre réel des messages envoyés par les implémentations. En conséquence, les verdicts émis par le système de test peuvent être biaisés. Pour résoudre les problèmes d'observation, ce rapport propose une méthode basée sur lŠestampillage logique: chaque sortie produite par les implémentations est associée à une horloge logique. En analysant lŠétiquette temporelle associée à chaque sortie, le testeur est capable de recréer la séquence réelle des événements observés afin de donner un verdict correct.*

Mots clés : *test d'interoperabilité, test passif, estampillage logique*

Travail pour la nouvelle collection de publications internes de l'Irisa version électronique compatible avec les RR Inria

* Projet Dionysos

1 Introduction

Modern networks are becoming heterogeneous and complex. As they are composed of equipments based on new technologies from a variety of vendors, it is frequent that errors would be introduced by differences in protocol implementations. Correct coordination and communication of different protocol implementations is essential to guarantee customer expectations.

To increase the confidence that network products conform to international standards, conformance testing methodologies have been standardized and deployed. Conformance testing [2] verifies whether a network component conforms to its specification. Although ISO-9646 standard mentions that conformance testing increases the probability of interoperability, it cannot be guaranteed. Conforming products may not interoperate due to several reasons: ambiguity in protocol standards, poorly specified protocol options, incompleteness of specifications, etc. Therefore, interoperability testing is indispensable to guarantee the correct operation of network products from different product lines and provide expected services.

To perform interoperability testing, the conventional method is the *active testing* approach [9, 5, 4]. Active testing is done by building a *test system* (TS) that is able to stimulate the *implementations under test* (IUT) and verify whether the outputs obtained for each stimulus are as expected. Generally, the test system tests the IUTs in a specific test environment. It sends carefully designed messages to the IUTs, receives what it responds and makes a conclusion. Although widely used, active testing has some limitations: Test can be difficult or even impossible to perform if the tester is not provided with a direct interface to stimulate the IUTs, or in operational environment where the normal operation of IUTs cannot be shutdown or interrupted for a long period of time.

Current trend in network components deployment is to shorten time-to-market and test their interoperability in operational environment. In this sense, active testing is not a suitable testing method especially for operational networks, where arbitrary inputs influences networks environment, sometimes may even provoke the crash of services.

The drawbacks of active testing lead to *passive testing* [7, 8]. Passive testing is a technique based on a high level of observation in order to establish a diagnostic regarding the implementations' behavior. Compared with active approach, one of the remarkable features of passive approach is that the test system does not interfere with the normal operation of the IUTs. Therefore, passive approach can be applied in operational networks, where it is usually difficult to insert an arbitrary input. Besides, passive approach makes it possible to carry out tests when the test system does not have direct access to stimulate the IUTs. In fact, the test can be done by only observing the trace produced by the IUTs with the help of network protocol analyzing tools such as *Wireshark*[1].

However, when passive interoperability testing is performed in operational environment, observation problems are often encountered due to the uncontrollable delay introduced by the asynchronous nature of the test environment. In fact, the observation order of the outputs produced by the implementations may be influenced. As a result, passive interoperability testing in such environment is error-prone: The verdict might be *permissive* – uncooperative IUTs are determined to be interoperable, or *biased* – the IUTs are determined non-interoperable but they are actually interoperable.

In this report, we propose a time-stamp-enabled mechanism that aims at solving the permissiveness and bias problems. The considered testing context is the passive *One-to-One* interoperability testing where there are two IUTs in interaction. The principle of the mechanism is to associate each event on IUTs with a logical time, which is piggybacked by the outputs sent by the IUTs. An algorithm is proposed to analyze the logical time and to recover the real occurrence order of the events that happened on each IUT. We prove that both bias and permissiveness of asynchronous interoperability testing can be suppressed by using this method. Interoperability verdicts remain the same in asynchronous test environment as in synchronous test environment.

The report is organized as follows: Section 2 represents the preliminaries. Section 3 presents the observation problems in asynchronous passive interoperability testing. Section 4 defines the time-stamp-enabled mechanism to solve the observation problems. The report is finished by a conclusion in section 5.

2 Preliminaries

2.1 General Interoperability Testing Architecture

The purpose of interoperability testing (*iop* for short in the sequel) is to verify n ($n \geqslant 2$) products from different product lines interoperate correctly and provide expected services. General interoperability testing architecture involves a *Test System (TS)* and a *System Under Test (SUT)* composed of n ($n \geqslant 2$) interconnected *Implementations Under Test (IUT)* from different product lines. According to the quantity of the IUTs, interoperability testing can be in either of the following context: *(i) Multi-Component* context where SUT has n ($n > 2$) IUTs. *(ii) One-to-One* context where SUT is composed of two IUTs. In practice, *One-to-One* iop is the most common context, either by testing the interoperability of IUTs in a pairwise way, or by testing the interoperability between one protocol implementation and a system already in operation (which may be composed of n IUTs).

[1] http://www.wireshark.org/

In interoperability testing, testing entities communicate with each other through a variety of interfaces:

- *Upper Implementation Access Point* ($UIAP_i$, $i = \{1, 2, ...n\}$) is the interface through which IUT_i communicates with its upper layer. UI_i is observable and controllable. Test system connected to $UIAP_i$ via *Upper Point of Control and Observation* ($UPCO_i$) interface can send test messages to the corresponding IUI_i and observe its output responses.
- *Lower Implementation Access Point* ($LIAP_i$) is the interface through which IUT_i communicates with its peer IUT. Contrary to $UIAP_i$, $LIAP_i$ is only observable. Test system connected to $LIAP_i$ via *Lower Point of Observation* (LPO_i) interface can only observe the interaction of IUT_i with its peer IUT but must not send any message to it.

2.2 Passive Interoperability Testing

In this report, we focus our work on *One-to-One passive interoperability testing* (cf. Fig.1)*:* The test between two IUTs is based only on observing their interaction (at $LIAP_i$ interfaces). No test message is allowed to be injected. The testing procedure consists of the following stages:

1. Interoperability test purpose (ITP) selection. The goal of testing is to find errors in SUT. However proving correctness is elusive as it is generally impossible to validate all possible behavior described in specifications. From the methodological point of view, most tests are carried on by setting test purposes. A test purpose is in general informal, in the form of an incomplete sequence of actions representing a critical property to be verified. It can be designed by experts or provided by standards guidelines for test selection [3].
2. Trace recording. Observable events (trace) produced by SUT are collected during the test execution.
3. Trace verification. The obtained trace is then analyzed against the interaction of IUTs' specifications to see whether the interoperability relation between the IUTs is satisfied. The execution trace verification returns a verdict : where $verdict \in \{Pass, Fail, Inconclusive\}$. $Pass$ means the test purpose is satisfied with no error detected. $Fail$ means at least an error is detected. $Inconclusive$ means the behavior of IUTs is correct w.r.t their specifications, however does not correspond to the test purpose.

In this report, we consider the asynchronous communication between testing entities: *(i)* The communication between two IUTs are asynchronous. In fact, protocol implementations interact by traversing several protocol layers. As delay is not controllable, it leads to asynchronous communications. *(ii)* The environment between the test system and the SUT is asynchronous too. In practice, passive interoperability testing is often done in a remote way, i.e., the test system reaches the SUT through a network. Accorindg to [1], asynchronous interaction can be modeled by two FIFO reliable queues, one for each direction between two testing entities and is denoted $\|_A$.(The environment between the TS and the SUT is modeled by two unidirectional FIFO queues as $LIAP_i$ interfaces are only observable). Moreover, we assume the *black box* testing context: The system has no knowledge of the IUTs inner structure. Only the external behavior of IUTs can be evaluated.

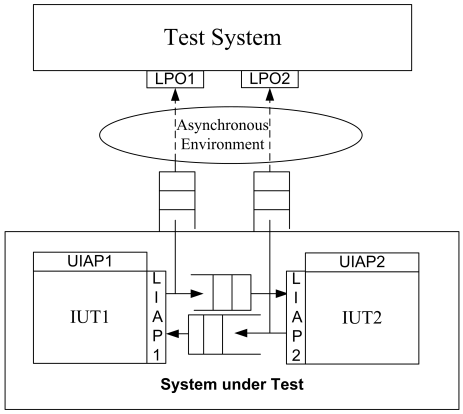

Figure 1: Passive interoperability testing architecture

2.3 Formal Model

In this report, the formal model used to describe specifications, implementations, etc is the IOLTS (Input-Output Labeled Transition System) model [1], which allows differentiating input, output and internal events while precisely indicating the interfaces specified for each event.

Definition 1 An IOLTS is a tuple $M = (Q^M, \Sigma^M, \Delta^M, q_0^M)$ where Q^M is the set of states of the system M with q_0^M its initial state. Σ^M is the set of observable events at the interfaces of M. Σ^M can be partitioned into: $\Sigma^M = \Sigma_U^M \bigcup \Sigma_L^M$, where Σ_U^M (resp. Σ_L^M) is the set of events at the upper (resp. lower) interfaces. Σ^M can also be partitioned to distinguish input (Σ_I^M) and output events (Σ_O^M). In IOLTS model, input and output actions are differentiated: We note $p?a$ (resp. $p!a$) for an input (resp. output) a at interface p. $\Delta^M \subseteq Q^M \times (\Sigma^M \cup \tau) \times Q^M$ is the transition relation, where $\tau \notin \Sigma^M$ stands for an internal action. Transition $(q, \alpha, q') \in \Delta^M$ can be equally noted $q \xrightarrow{\alpha}_M q'$.

Other notations Let us consider an IOLTS M, and let an observable event $\alpha \in \Sigma^M$ with $\alpha = p \cdot \{?, !\} \cdot m$, an executable event of the system $\mu_i \in \Sigma^M \cup \tau$, a succession of events $\sigma \in (\Sigma^M)^*$, and q, q', $q_i \in Q^M$. We define the following notations:

- $q \xrightarrow{\mu_1 \cdots \mu_n}_M q' =_{def} \exists q_0 = q, q_1, \ldots, q_n = q'. \forall i \in [1, n], q_{i-1} \xrightarrow{\mu_i}_M q_i$.
- $q \overset{\epsilon}{\Rightarrow}_M q' =_{def} q = q'$or $q \xrightarrow{\tau \ldots \tau}_M q'$.
- $Out(q) =_{def} \{\alpha \in \Sigma_O^M | \exists q', q \xrightarrow{\alpha}_M q'\}$ is the set of possible outputs at state q. Similarly, $In(q) =_{def} \{\alpha \in \Sigma_I^M | \exists q', q \xrightarrow{\alpha}_M q'\}$ is the set of possible inputs and $\Gamma(q) =_{def} \{\alpha \in \Sigma^M | \exists q', q \xrightarrow{\alpha}_M q'\}$ is the set of all possible events at the state q.
- q after $\sigma =_{def} \{q' \in Q^M | q \overset{\sigma}{\Rightarrow}_M q'\}$ is the set of states which can be reached from q by the sequence of actions σ. By extension, all the states reached from the initial state of the IOLTS M are (q_0^M after σ) and will be noted by(M after σ).
- $\bar{\mu} = p!a$ if $\mu = p?a$ and $\bar{\mu} = p?a$ if $\mu = p!a$. For internal events, $\bar{\tau} = \tau$.

2.4 Passive Interoperability Criteria

Passive One-to-One interoperability criteria are defined in [9] as follows:

Definition 2 In passive One-to-One interoperability testing, two IUTs are considered interoperable I_1 *iop* $I_2 =_{def}$

1. After a trace during the asynchronous interaction of IUTs, all the outputs observed at the interfaces of IUTs must be foreseen in the interaction of their specifications.
   $\forall\, \sigma \in Trace(S_1 ||_A S_2) \Rightarrow Out((I_1 ||_A I_2), \sigma) \subseteq Out((S_1 ||_A S_2), \sigma)$.
2. And, all messages sent by one IUT to the other IUT must be received by the latter. It should be mentioned that interoperability testing is a black box testing. The TS is able to observe a message which has been sent by an IUT to the other IUT via its interface, but not whether the latter has actually processed the message or not. Therefore, the verification of an input is in fact done by verifying its causal-dependent observable outputs. To confirm that an input has been received involves having verified its related outputs that cannot be observed unless the input has been executed.
   $\forall \{i, j\} = \{1, 2\}, i \neq j; \forall \sigma \in Trace(S_i ||_A S_j), \sigma_i \in Traces(S_i), \sigma_j \in Traces(S_j), \forall \mu \in Out(I_i, \sigma_i), \forall \sigma' \in ((\Sigma^{S_i} \cup \Sigma^{S_j}) \backslash \overline{\mu})^*,$
   $\sigma.\mu.\sigma'.\overline{\mu} \in Traces(S_i ||_A S_j) \Rightarrow Out(I_j, \sigma_j.\sigma'.\overline{\mu}.\sigma_k) \subseteq Out(S_j, \sigma_j.\sigma', \overline{\mu})$ where $\sigma_k \in (\Sigma_I^{S_j})^*$.

3 Observation Issues in Remote Passive Interoperability Testing

As mentioned in Section 2, passive interoperability testing is done by building a test system that observes the interaction between two IUTs. Within sight of the observations, a verdict (Fail, Pass or Inconclusive) is emitted w.r.t the passive interoperability criteria (cf. Definition 2). In fact, the passive interoperability cirteria considers only the asynchronous interaction of IUTs. In practice however, one cannot always neglect the test environment intercalated between the test system and the SUT. The most common situation is that the TS reaches the SUT through a network connection (remote testing). In such case, the delay in network causes asynchronism and should be taken into account as well.

In remote interoperability testing, the asynchronous nature of the environment often results in observation problems: The delay in environment influences the observation order of the events produced by IUTs. There are generally two kinds of problems: *(i)* the problem of permissiveness that happens when uncooperative IUTs are judged to be interoperable. *(ii)* the problem of bias occurs when network components are determined uncooperative but they are actually interoperable. The verdict of asynchronous interoperability testing is error-prone.

Example 1 Let us consider the specifications S_1 and S_2 in Fig.2, which represent a connection protocol: Initially, S_1 sends a request ($1!cnr$) for connection. After S_2 receives the request ($2?cnr$), it can either accept the connection ($2!ack-cnr$) or decline it ($2!nack-cnr$). If S_2 accepts the connection, the connection will be established by an acknowlegement of S_1 ($1!ACK$). Then, both S_1 and S_2 can terminate the connection by sending Fin request, which should be followed by an $ack-Fin$ acknowlegement.

Fig.3 illustrates three implementations based on the specifications: I_1=S_1, I_2 and I_3 are two implementations based on S_2. This example shows that passive interoperability testing carried out in asynchronous test environment introduces permissiveness and bias observation issues and causes biased verdicts.

We use I_i iop_A I_j to denote the interoperability relation of IUT_i and IUT_j in asynchronous test environment. We show that I_i iop $I_j \not\Leftrightarrow$ I_i iop_A I_j .

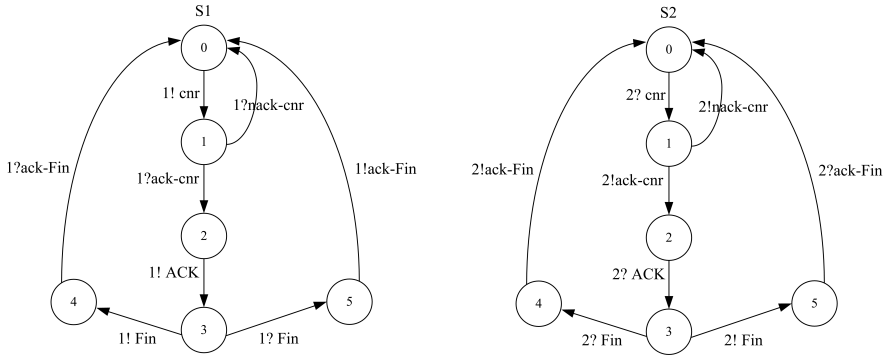

Figure 2: Specifications S_1 and S_2

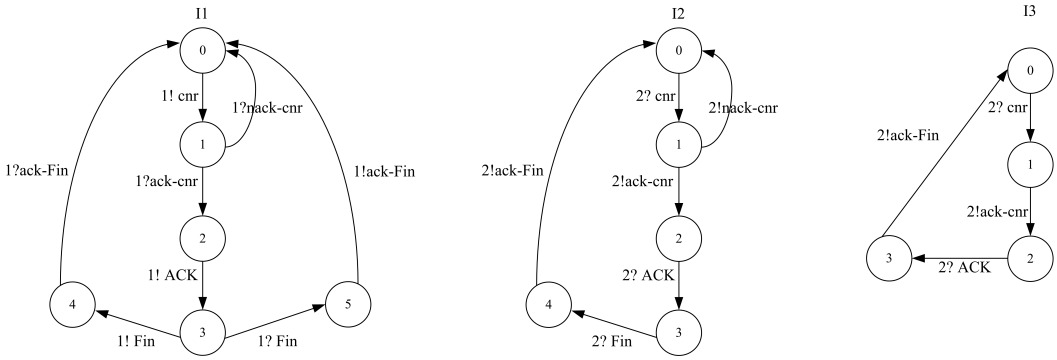

Figure 3: Implementation I_1, I_2 and I_3

- *Permissiveness*: passive interoperability testing in remote asynchronous environment is more permissive. The problem is brought to light by the interaction between I_1 and I_3 (cf. Fig.4). According to the passive interoperability criteria (cf. Definition 2), I_1 and I_3 do not satisfy interoperability relation.
  We have $\neg(I_1$ iop $I_3)$ due to the output $2!ack-Fin$ produced by I_3 after the input $2?Ack$ specified in S_2. However, as mentioned before, the verification of an input in a black box testing context is based on verifying its causal dependent observable outputs. Here, the input $2?Ack$ is done by verifying its causal dependent events $1!Ack \rightarrow 2!Fin$. Therefore, as illustrated in Fig.4[2], $Out(I_1||_A I_3, 1!Ack) \subseteq Out(S_1||_A S_2, 1!Ack) = 2!Fin$, thus the I_1 and I_3 are determined interoperable. In fact, the observation of TS does not reveal the real order of the events that happened on each IUT.

[2] $P1?cnr =_{def} PO_1$ observes output cnr sent by IUT_1. The rest may be deduced by analogy.

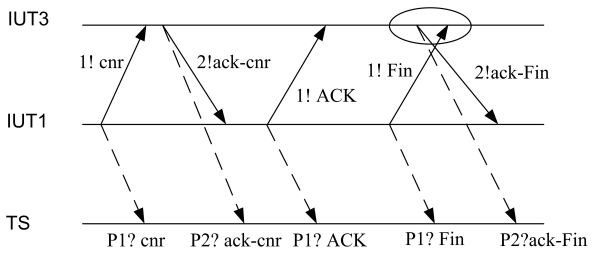

Figure 4: Permissive verdict

- *Bias*: This problem can be illustrated by the interaction of I_1 and I_2 (cf. Fig.5). According to the passive interoperability criterion, $I_1 iop I_2$. However, the delay in asynchronous testing environment intercalated between the TS and SUT can cause the disorder of the output observation. As a result, the trace $1!Fin$, $2!ack-Fin$ can be observed as $2!ack-Fin$, $1!Fin$ from the point of view of the TS. According to the specifications, $Out(S_1||_A S_2, 2!ack-Fin)= 1!cnr$ but we have here $Out(I_1||_A I_2, 2!ack-Fin)= 1!Fin \notin Out(S_1||_A S_2, 2!ack-Fin)$. Thus $\neg(I_1 iop_A I_2)$.

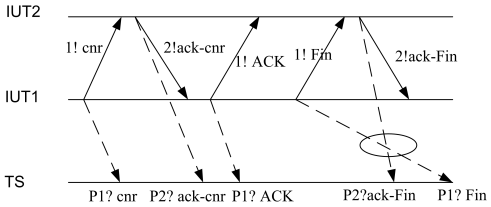

Figure 5: Biased verdict

To sum up, in remote passive interoperability testing, observation problems maybe encountered due to the delay in asynchronous testing environment [10], [11]. In the next section, we show that these problems can be avoided by an appropriate time stamp enabled mechanism.

4 Dealing with Observation Problems in Passive Interoperability Testing

In this section, the main issue is to keep a high level of observation in passive interoperability testing activity, ensuring that no biased verdict is introduced while executing the test in an asynchronous environment. The principle is to instrument each IUT by using a logical time enabled mechanism [6], [12] so that each output from IUTs brings additional information about the order in which each event is produced.

The method is illustrated in Fig.6. The idea is to implement a counting mechanism ST on the synchronous parallel composition of each IUT. The role of the ST is to code the occurrence of the events that happen on each IUT by time stamps. Then, each time an output is produced, its corresponding time stamp is piggybacked. A *log* file is used to store all the observed outputs from each IUT as well as their time stamps. At the side of the tester, another system $\overline{ST}$ is implemented. The role of $\overline{ST}$ is to reorder all the events in *log* by analyzing their associated time stamps. We will show that $\overline{ST}$ is able to reconstruct all the events that happen on each IUT with the help of time stamps. Finally, these well ordered traces will be analyzed by the TS against the specifications to give a diagnose.

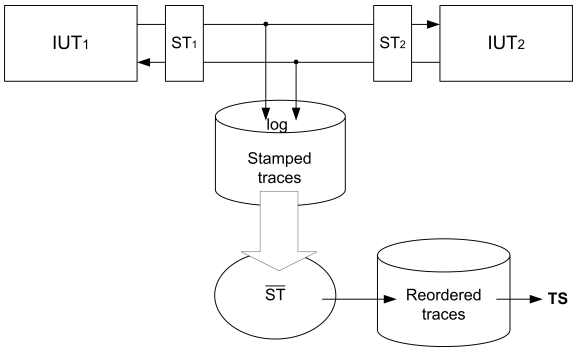

Figure 6: Time stamp enabled asynchronism management mechanism

4.1 Counting Mechanism ST

Fig.7 illustrates in detail the logical time mechanism proposed in this report: The counting mechanism ST implements a counter CT_i on the corresponding IUT_i (i={1,2}). The start and the end of trace recording are managed by TS at instant t and $t+\delta$ respectively.

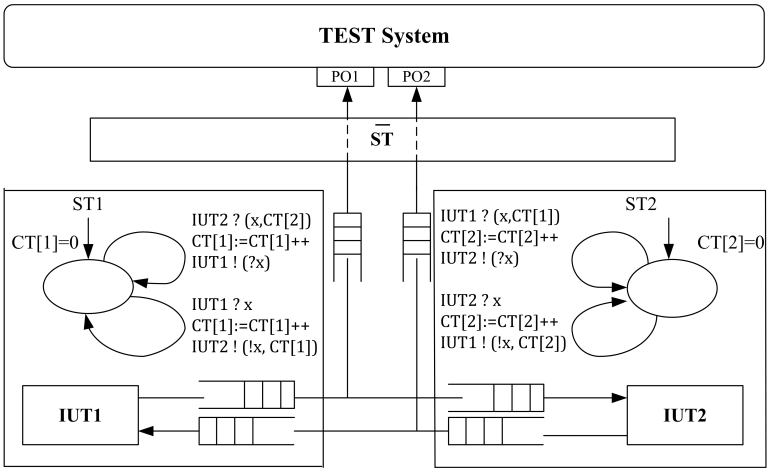

Figure 7: Implementation of time stamp mechanism

- Initially at instant t, $\forall\{i,j\}$ ={1,2} ,$i \neq j$; $CT_i = CT_j = 0$.
- From t to $t+\delta$
  - each time IUT_i sends a message α to IUT_j, ST_i increments CT_i by 1. Message α and its time stamp, denoted by a tuple (α, CT_i) are then sent to IUT_j (cf. Fig.7). During message transmission, (α, CT_i) is observed by PO_i and put into *log*.
  - Each time ST_i receives a tuple (α, CT_j) from IUT_j, ST_i increments CT_i by 1. Then CT_j is removed, only message α will be sent to IUT_i.

In fact, CT_i is a counter that associates the occurrence of each event on IUT_i with a time stamp. Thus, CT_i reveals the sequence number of an event that happens on IUT_i.

4.2 Time Stamp Analysis

All the observed outputs from both IUTs as well as their associated time stamps are stored in *log*. The time stamp information is then analyzed by the TS.

At the side of TS, an algorithm $\overline{ST}$ is implemented. We call it $\overline{ST}$, the opposition transformation of ST since an output of IUT_i corresponds to an input of PO_i. The role of $\overline{ST}$ is to analyze the logical stamps associated to each message recorded in *log* when trace registration is finished, and recovers the occurrence order of the events that happened on each IUT. $\overline{ST}$ involves two stages:

1. The construction of MQ (*Mirror Event Queue*, cf. Definition 3), which aims at finding the non-directly observable inputs on each IUT and helps reconstructing the occurrence order of the events that happened on each IUT.
2. The construction of OQ (*Ordering Queue* cf. Definition 4), which intends to recover the occurrence order of the events that happened on each IUT.

Definition 3: MQ_i (The Mirror Event Queue of IUT_i). MQ_i is constructed recursively by checking each output in *log* one by one. Formally, $\forall\{i,j\}$ ={1,2} ,$i \neq j$; $\forall\sigma \in log$, $\alpha \in \Sigma_O^{IUT_i}$, $\beta \in \Sigma_O^{IUT_j}$: $MQ_i(\epsilon)=\epsilon$, $MQ_i(\sigma.\alpha)= MQ_i(\sigma)$, $MQ_i(\sigma.\beta) = MQ_i(\sigma).i?\beta$.
In fact, for all the observed outputs α produced by IUT_i taken in order from *log*, their mirror events $\overline{\alpha}$ will be put into the mirror event queue MQ_j of IUT_j with respect to their observation order. As the channel between each testing entity is FIFO reliable, the sequence in MQ_j will be received in this order by IUT_j.
For example, for a trace $\sigma \in log$: $(1!cnr, 1)$, $(2!ack-cnr, 2)$, $(1!ACK, 3)$, $(1!Fin, 4)$, $(2!ack-Fin)$. The corresponding *Mirror Event Queues* are such that MQ_1= $1?ack-cnr$, $1?ack-Fin$; MQ_2= $2?cnr$, $2?ACK$, $2?Fin$.

Definition 4: OQ_i: The Ordering Queue of IUT_i. OQ_i stores the events reordered by $\overline{ST}$ on IUT_i .

The construction of OQ is in fact the trace ordering procedure which involves in checking each tuple (α, CT_i) taken in order from the *log*. It is composed of the following steps:

1. For each tuple (α, CT_i) where $\alpha \in \Sigma_O^{IUT_i}$, $\overline{ST}$ carries out the calculation of $X = CT_i - 1 - length(OQ_i)$, where $length(OQ_i)$ is equal to the number of elements in OQ_i.

   Proposition 1 $\forall(\alpha \in \Sigma_O^{IUT_i}, CT_i)$, $X = CT_i - 1 - length(OQ_i)$ is the number of unknown events that happened before α.

   Proof: CT_i presents the sequence number of the output α on IUT_i. Thus $CT_i - 1$ states for the number of the events that happened before α. As OQ_i stores the events already ordered on IUT_i, the number of unknown events before α is therefore $CT_i - 1 - length(OQ_i)$.

   Proposition 2 Unknown events on IUT_i before an output α are input events.

   Proof: As the test environment is modeled as reliable FIFO queues, the observation of the outputs produced by the same IUT repect the FIFO order as well. Therefore, the unknown events before α on IUT_i can only be inputs.

2. According to the value X obtained by step 1, different actions can be taken:

- If $X = 0$, α is added to the tail of OQ_i. i.e., there is no unknown event before α on IUT_i, α can be directly put into its ordering queue.
- If $X > 0$, i.e., there are unknown event(s) before α. According to Proposition 2, these unknown events can only be inputs. Therefore, $\overline{ST}$ looks into the Mirror Event Queue of IUT_i MQ_i, as MQ_i stores the mirror events of observed outputs sent by IUT_j, which will be received by IUT_i in a FIFO order.
  MQ_i can be decoupled to two parts: $MQ_i = MQ_i^{'}.\ MQ_i^{''}$ where $MQ_i^{'}$ represents the unknown inputs before α, and $length(MQ_i^{'}) = X$. Then, OQ_i is updated: $MQ_i^{'}$ is taken out from MQ_i and added to the tail of OQ_i, so as α.

The algorithm $\overline{ST}$ is written below formally:

Algorithm 1: Algorithm $\overline{ST}$

Input: *log*
Output: OQ_i, OQ_j
/*reordered events on both IUTs*/
Initialization: $OQ_i = OQ_j = MQ_i = MQ_j = \epsilon$, $\forall\{i,j\} = \{1,2\}$,$i \neq j$;
Begin
/*The construction of MQ*/
for *each* $\alpha \in$ log **do**
  if $\alpha \in \Sigma_O^{IUT_i}$ **then**
    $MQ_j = MQ_j.j?\alpha$;
  end
end
/*The construction of OQ*/
for *each* $\alpha \in$ log **do**
  if $\alpha \in \Sigma_O^{IUT_i}$ **then**
    $X = CT_i - 1 - length(OQ_i)$;
    Let $MQ_i^{'}.MQ_i^{''}$ s.t. length $(OQ_i.MQ_i^{'}) = CT_i - 1$ and $MQ_i = MQ_i^{'}.MQ_i^{''}$
    $OQ_i = OQ_i.MQ_i^{'}.\ \alpha$;
    $MQ_i = MQ_i^{''}$;
  end
end
Return OQ_i, OQ_j
End

Complexity of the algorithm The algorithm is composed of two *for* loops since *log* is examined twice for the construction of MQ and OQ respectively. For the first *for* loop, the complexity is $O(n)$ where n is the size of *log*. The same, the second *for* loop has the complexity of $O(n)$ as well. Therefore, the whole algorithm has the complexity of $O(n)$.

Proposition 3: For a trace σ produced on IUT_i in asynchronous environment, the application of algorithm $\overline{ST}$ reconstructs σ.

Proof: First, the proposed time-stamp-enabled mechanism associates each event that happens on IUT_i with a time stamp. This time stamp information is carried on-the-fly with the outputs and is analyzed by $\overline{ST}$. Although asynchronous environment disturbs the observation order, it preserves the relative order of inputs and outputs on each IUT. Therefore, logical time can be used to reconstruct σ.

Theorem 1: By using the proposed time-stamp-enabled mechanism, permissive and biased verdicts are avoided.

Proof: We first prove that $I_1\ iop_A\ I_2 \Rightarrow I_1\ iop\ I_2$ by using the proposed time stamp enabled mechanism.

First, the verdict $I_1\ iop_A\ I_2$ after using the proposed mechanism algorithm $\overline{ST}$ implies $\exists\ \overline{ST}(\sigma) \in Trace(S_1||_A S_2)$, $Out((I_1||_A I_2), \overline{ST}(\sigma)) \subseteq Out((S_1||_A S_2), \overline{ST}(\sigma))$. As proposition 3 said that $\overline{ST}$ can reconstruct σ. Therefore, $Out((I_1||_A I_2), \sigma) \subseteq Out((S_1\ ||_A\ S_2), \sigma)$. Moreover, as $\overline{ST}(\sigma)$ reconstructs the occurrence order of both inputs and outputs. $I_1\ iop_A\ I_2 \Rightarrow Out(I_j, \sigma_j.\sigma^{'}.\overline{\mu}.\sigma_k) \subseteq Out\ (S_j, \sigma_j.\sigma^{'}, \overline{\mu})$ where $\sigma_k \in (\Sigma_I^{S_j})^*$ Therefore, $I_1\ iop_A\ I_2 \Rightarrow I_1 iop I_2$.

Then we prove that $\neg(I_1\ iop_A\ I_2) \Rightarrow \neg(I_1 iop I_2)$. It is trivial since it is the converse of $I_1\ iop_A\ I_2 \Rightarrow I_1 iop I_2$.

Therefore, passive interoperability verdicts remain non-permissive and unbiased in asynchronous environment by using the proposed time stamp enabled mechanism.

Let's review the example of Fig.4 and Fig.5 to see how permissiveness and bias issues can be suppressed by using the proposed time stamp enabled mechanism.

Example 2 : The suppression of permissiveness is illustrated in Fig. 8. Each event is associated with a time stamp. Time stamps are carried on-the-fly with the outputs sent by each IUT. The process of time stamp decoding performed by $\overline{ST}$ is the following:

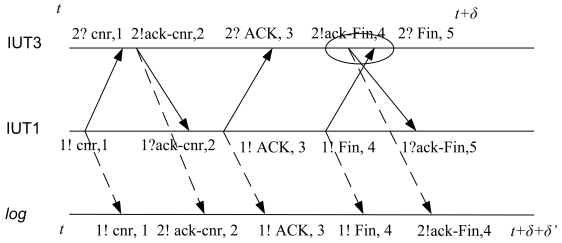

Figure 8: The Suppression of Permissiveness

1. The construction of MQ from *log*.
   (a) MQ_1= $1?ack - cnr$, $1?ack - Fin$.
   (b) MQ_2= $2?cnr$, $2?ACK$, $2?Fin$.
2. The construction of OQ by analyzing each event in *log*.
   (a) After analyzing$(1!cnr, 1)$,OQ_1 =$1!cnr.$; MQ_1= $1?ack - cnr$, $1?ack - Fin$.
   (b) After analyzing$(2!ack - cnr, 2)$,OQ_2 =$2?cnr$, $2!ack - cnr.$; MQ_2=$2?ACK$, $2?Fin$.
   (c) After analyzing$(1!ACK, 3)$, OQ_1 =$1!cnr$, $1?ack - cnr$, $1!ACK.$; MQ_1=$1?ack - Fin$.
   (d) After analyzing$(1!Fin, 4)$,OQ_1 =$1!cnr$, $1?ack - cnr$, $1!ACK$, $1!Fin.$;MQ_1=$1?ack - Fin$.
   (e) After analyzing $(2!ack - Fin, 4)$,OQ_2 =$2?cnr$, $2!ack - cnr$, $2?ACK$, $2!ack - Fin.$; MQ_2=$2?Fin$.

   Therefore, by comparing with the specification of S_1 and S_2, TS detects that IUT_3 has chosen a different behavior from the S_2 by sending $2!ack - Fin$ before receiving $2?Fin$. $\neg(I_1\ iop_A\ I_3)$. Permissiveness is avoided.

Similarly, bias problem can be avoided by using the method.

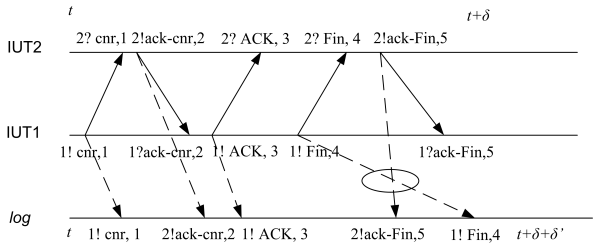

Figure 9: The Suppression of Bias Problem

After examining *log*, TS gets

1. OQ_1 =1!cnr, 1?$ack-cnr$, 1!ACK, 1!Fin.
2. OQ_2 =2?cnr, 2!$ack-cnr$, 2?ACK, 2?Fin, 2!$ack-Fin$.

Which are correct w.r.t the interaction of their specifications. Therefore I_1 $iop_A I_2$, bias is avoided.

To sum up, to solve the observation problems cause by asynchronous test environment in passive interoperability testing methods, the idea is to implement a counting mechanism ST on the synchronous parallel composition of each IUT. Then, from an event trace, a kind of test driver implementing $\overline{ST}$ can reconstruct the sequence of events which occurred on both IUTs. These sequences will be checked against the specifications to decide whether they satisfy the interoperability relation.

5 Conclusion and Future Work

In this report, a stamped interoperability testing method has been proposed to solve the observation problems in asynchronous One-to-One passive interoperability testing. By using this method, both permissiveness and bias observation issues can be avoided. The test system can reconstruct a correct trace without out-of-sequence event produced by the IUTs.

To obtain a correctly ordered trace is essential because no matter which method is used to carry out interoperability, a correct observation of the interaction of the IUTs is always necessitated. It will help the test system give a correct verdict. Normally, the trace will be analyzed so as to decide whether the IUTs are interoperable or not.

The solution proposed for One-to-One passive interoperability testing can be extended to other more complex situation such as multi-components context where more than two IUTs interact. The main difficulty for multi-components context will be managing the causal relationship between inputs and outputs since several inputs can arrive simultaneously to one IUT. Future work will focus on researching the asynchronous multi-components context and solving observation issues in this context.

References

[1] L.Verhaard, J.Tretmans, P.Kars, and E.Brinksma. On asynchronous testing. In Gregor von Bochmann, Rachida Dssouli, and Anindya Das, editors, Protocol Test Systems, volume C-11 of IFIP Transactions, pages 55-66. North-Holland, 1992.

[2] ISO. Information Technology-open system interconnection Conformance Testing methodology and framework-Parts 1-7. International Standard ISO/IEC 9646/1-7,1994.

[3] S.Schulz, A. Wiles, and S. Randall. TPLan - A notation for expressing test purposes. ETSI, TestCom/FATES, LNCS 4581, 292–304, 2007.

[4] R.Hao, D.Lee, R.K.Sinha, and N.Griffeth. Intergated system interoperability testing with applications to VoIP. IEEE/ACM Trans. Netw., 12(5): 823-826, 2004.

[5] S.Seol, M. Kim, S.Kang, and J.Ryu. Fully automated interoperability test suite derivation for communication protocols. Computer Networks, 43(6): 735-759, 2003.

[6] R.Baldoni and M.Raynal. Fundamentals of Distributed Computing. IEEE Distributed Systems Online, 3(2):1-18, 2002.

[7] F.Zaidi, A.Cavalli, and E.Bayse. NetworkProtocol Interoperability Testing based on Contextual Signatures. The 24th Annual ACM Symposium on Applied Computing SAC'09, Hawaii, USA, 2009.

[8] D.Lee, A.N.Netravali, K.K.Sabnani, B.Sugla, and A.John. Passive testing and applications to network management. In International Conference on Network Protocols, ICNP'97, pages 113-122. IEEE Computer Society Press, 1997.

[9] A.Desmoulin. Test d'interopérabilité de protocoles: la formalisation des critères d'interopérabilité la génération des tests. Doctor thesis. University Rennes 1, 2007.

[10] L. Doldi, V.Encontre, J.C.Fernandez, T.Jéron, S.Le Bricquir, N. Texier, and M.Phalippou. Assessment of automatic generation methods of conformance test suites in an industrial context. In B.Baumgarten, H.J.Burkhardt, and A.Giessler, editors, IFIP TC6 9th International Workshop on Testing of Communicating Systems. Chapman & Hall, pages 346-361, 1996.

[11] C.Jard, T.Jéron, L.Tanguy, and C.Viho. Remote testing can be as powerful as local testing. FORTE/PSTV'99, Joint International Conference on formal description techniques and protocol specification testing and verification, Beijing, 1999.

[12] L.Lamport. Time, clocks, and the ordering of events in a distributed system. Communications of the ACM Volume 21 (7): 558–565,1978.